University of Reading

CDASH: a cloud-enabled program for structure solution from powder diffraction data

Article

Accepted Version

Spillman, M. J., Shankland, K., Williams, A. C. and Cole, J. C. (2015) CDASH: a cloud-enabled program for structure solution from powder diffraction data. Journal of Applied Crystallography, 48 (6). pp. 2033-2039. ISSN 0021-8898 doi: https://doi.org/10.1107/S160057671502049X Available at https://centaur.reading.ac.uk/48145/

It is advisable to refer to the publisher's version if you intend to cite from the work. See Guidance on citing.
Published version at: http://scripts.iucr.org/cgi-bin/paper?S160057671502049X
To link to this article DOI: http://dx.doi.org/10.1107/S160057671502049X

Publisher: Wiley-Blackwell Publishing, Inc

All outputs in CentAUR are protected by Intellectual Property Rights law, including copyright law. Copyright and IPR is retained by the creators or other copyright holders. Terms and conditions for use of this material are defined in the End User Agreement.

www.reading.ac.uk/centaur

CentAUR

University of Reading

Central Archive at the University of Reading

Reading's research outputs online

CDASH: a cloud-enabled program for structure solution from powder diffraction data

Authors

Mark J. Spillman[a]*, Kenneth Shankland[a], Adrian C. Williams[a] and Jason C. Cole[b]

[a]School of Chemistry, Food and Pharmacy, University of Reading, Whiteknights, Reading, Berkshire, RG6 6AP, UK

[b]Cambridge Crystallographic Data Centre, 12 Union Road, Cambridge, CB2 1EZ, United Kingdom

Correspondence email: m.j.spillman@reading.ac.uk

Synopsis A cloud-computing approach to accelerating structure determination from powder diffraction data, based on the Amazon EC2 system and the *DASH* software, is presented.

Abstract The simulated annealing approach to crystal structure determination from powder diffraction data, as implemented in the *DASH* program, is readily amenable to parallelisation at the individual run level. Very large scale increases in speed of execution can be achieved by distributing individual *DASH* runs over a network of computers. The *CDASH* program delivers this by using scalable, on-demand computing clusters built on the Amazon Elastic Compute Cloud service. By way of example, a 360 vCPU cluster returned the crystal structure of racemic ornidazole (Z'=3, 30 degrees of freedom) *ca.* 40 times faster than a typical modern quad-core desktop CPU. Whilst used here specifically for *DASH*, this approach is of general applicability to other packages that are amenable to coarse-grained parallelism strategies.

1. Introduction

DASH (David et al., 2006; David et al., 1998), a computer program for crystal structure determination from powder diffraction data (SDPD) that utilises a simulated annealing (SA) algorithm, has previously been adapted to run on computers that have multiple CPU-cores *via MDASH* (Griffin *et al.*, 2009b) and distributed computing systems *via GDASH* (Griffin *et al.*, 2009a). Another SDPD package, *FOX* (Favre-Nicolin & Cerny, 2002), has also been adapted to utilise multi-core architectures and distributed computing, *via* the *FOX.Grid* add-on (Rohlíček *et al.*, 2007).

Whilst the distributed computing capability provided by *GDASH* and *FOX.Grid* allow for orders-of-magnitude increases in the computing power that can be brought to bear on a SDPD task, the distributed computing approach to SDPD has seen only limited use. In the case of *GDASH*, this is

IMPORTANT: this document contains embedded data - to preserve data integrity, please ensure where possible that the IUCr Word tools (available from http://journals.iucr.org/services/docxtemplate/) are installed when editing this document.

almost certainly because of the significant financial hurdle presented by the specialised grid management software required and its associated maintenance.

This work demonstrates how the distributed computing concept used by *GDASH* can be realised using cloud-computing services, which remove the necessity for users to have a large existing network of 'in-house' computers.

Cloud-computing is a broad term that covers a variety of activities. In this work, we make use of *infrastructure as a service* (IaaS) type cloud computing. IaaS enables users to access a wide range of computers (usually in the form of virtual machines) over their internet connection. These machines can then be used for any computing task required. One popular use of IaaS resources is on-demand, high-performance computing. One well known IaaS provider is the Amazon Elastic Compute Cloud (EC2) (Amazon, 2010) that permits on-demand creation of a wide variety of virtual machines[1], which when running are known as *instances*. The latest generation of high-performance computing instances (at time of writing, those with prefix c4[2]) are listed in Table 1. Full details of all available instance types are available on the Amazon Web Services (AWS) website (Amazon, 2014). The EC2 service has previously been utilised in a crystallographic context (de Oliveira *et al.*, 2011) and for processing single particle cryo-electron microscopy data (Cianfrocco & Leschziner, 2015).

EC2 instances use customisable operating systems, known as Amazon Machine Images (AMIs). These can be pre-installed with software suited to diverse purposes such as high performance computing, web servers, database management, video rendering and application development. Once a suitable AMI has been created, multiple instances of that AMI can be started whenever required. Microsoft Windows, Linux and BSD operating systems are supported.

Using the instance types available, bespoke computing clusters can be created without the need to invest in hardware, which is of course subject to depreciation and obsolescence. Amazon provides an application programming interface (API) which allows third-party tools to create, manage and interact with instances, but writing software at the API level for cluster creation and management is a time consuming task. Fortunately, toolkits such as StarCluster (Section 1.1) exist, which provide convenient and easy-to-use interfaces for cluster creation, control and management.

1.1. StarCluster

[1] For the purposes of this paper, a virtual machine can be defined as an operating system that is installed onto software which imitates dedicated hardware. For example, a virtual machine running Linux can be created on a computer running MS Windows as its base operating system, provided the appropriate virtualisation software is installed on Windows. The end user interacts with Linux virtual machine exactly as if it were installed on the underlying hardware. Importantly, the virtual machines can be created and destroyed at will.

[2] According to the AWS documentation, *"C4 instances are based on custom 2.9 GHz Intel® Xeon® E5-2666 v3 (Haswell) processors, optimized specifically for Amazon EC2"* - http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/c4-instances.html

StarCluster (STAR, 2014) is an open source toolkit used to automate the building, configuration and management of high performance Linux-based compute-clusters on the EC2 service. The clusters it builds are configured with one instance serving as the *master node* for the cluster, whilst any remaining instances serve as *worker nodes* as shown in Figure 1. As such, users primarily interact with the master node of the cluster, which then distributes jobs to the workers. By default, StarCluster also executes jobs on the master node of the cluster, maximising utilisation of the computing resources available. In this work, it is used as the basis for *CDASH* due to its ease of use and the rich set of features it provides.

2. *CDASH* overview

CDASH is a lightweight command-line driven program written in the Python programming language. It takes, as input, files generated by *DASH* and the parameters (defined by the number of instances and instance type required) of the bespoke cluster requested by the user. It then automatically creates a cluster of the requested specification on EC2, uploads files, queues jobs to run, checks job completion status, downloads results and terminates the cluster. The results returned by *CDASH* are standard *.dash* result files, which can be viewed, merged and manipulated locally as normal. Figure 2 provides a schematic of the *CDASH* mode of operation.

CDASH is intended to accelerate the structure determination of crystal structures that currently require a few days or more of local CPU time to solve; these tend to be examples with large volume asymmetric units, $Z' > 1$ and more than 30 degrees of freedom.

3. Program description

3.1. Running *CDASH*

3.1.1. Software requirements and AMI preparation

The following discussion assumes that the user already has an Amazon Web Services (AWS) account[3]. Table 2 lists the software prerequisites for *CDASH* use, including both local software and the packages that must be present on the AMI in order to allow *CDASH* to run. Due to these requirements, prior to using *CDASH* for the first time, a user must prepare a custom Linux-based AMI that has the software pre-requisites listed in Table 2 installed, as well as a copy of *DASH* that has been installed and correctly configured[4]. This process is straightforward, with detailed instructions available in the *CDASH* documentation. To ensure maximum compatibility with StarCluster, it is strongly recommended that the custom AMI be based on a StarCluster public AMI. If this is not done,

[3] AWS accounts can be created for free by following this link: http://aws.amazon.com/
[4] *DASH* installation on a Linux-based AMI is accomplished using the Wine compatibility layer software. This means that the *DASH* executable used on the cloud is identical to the one used locally.

additional packages must be installed on the AMI in order for StarCluster to work[5]. Once the prerequisites listed in Table 2 have been installed, the user saves the AMI using the tools provided by AWS and configures *CDASH* to use that AMI for future clusters. Note that we utilise the Wine compatibility layer (Wine Project, 2015) in order to let us install and run *DASH* (which normally runs in an MS Windows environment) on a Linux virtual machine. This layer has a negligible impact on *DASH* run times and removes the requirement for a Linux-specific version of *DASH* to be compiled.

3.1.2. Preparation of input files

The procedure for PXRD data preparation and *DASH* batch file (DBF) generation on the local computer is identical to the procedure used for *MDASH* and *GDASH*, and is described elsewhere in detail (Griffin *et al.*, 2009b, a). The *DASH* batch files (DBFs), which have the extension *.dbf*, are saved in the same directory as the files generated in the *DASH* Pawley fitting procedure (files with extensions *.sdi*, *.hcv*, *.tic*, *.dsl* and *.pik*) together with the *.zmatrix* representations of the structural fragments. These files are henceforth referred to as the "*DASH* files", and the folder containing them the "working directory".

3.1.3. Command line invocation

After opening a command window on the local computer and navigating to the working directory, *CDASH* is invoked using the command:

```
python cdash.py
```

The user is then prompted to specify which instance type to use, and the number of instances required for the cluster. Alternatively, these parameters may be supplied as command line arguments as follows:

```
python cdash.py –i instance-type –n N
```

where the argument *instance-type* is a string indicating the type of instance requested (typically one from Table 1) and the argument *N* is an integer that determines the number of instances of type *instance-type* to be included in the cluster. The user is prompted for these parameters if they are omitted.

The appropriate choice of instance type and cluster size is dependent on the number of SA runs to be performed, and the approximate duration of each SA run. AWS charges for instance time on an hourly basis; a job lasting 1 minute and a job lasting 59 minutes would both be charged at a full hour. As such, it is recommended that for a given number of SA runs, the number of vCPUs in the cluster should be tailored to maximise the computing resources that are being paid for. By way of example, a set of 100 SA runs, each taking 10 minutes could be processed on a cluster with 100 vCPUs or a

[5] This process lies outside the scope of this publication, though instructions are readily available online.

cluster with 25 CPUs. Ignoring cluster start-up and network transfer overheads, the cluster with 100 vCPUs will complete the allotted runs in approximately 10 minutes whilst the 25 vCPU cluster will complete the runs in approximately 40 minutes. The latter scenario, whilst slower, will be cheaper. When embarking on a structure determination that is likely to involve very large numbers of SA runs, it is advisable to do a small "benchmarking" CDASH run, to aid in selecting the optimal instance type and cluster size.

Beyond this point, *CDASH* requires no further user interaction: it automatically detects the *DASH* files in the working directory and creates a cluster based on the user-defined instance type and size. Once the cluster is running, files are uploaded and distributed around the cluster, jobs are queued and tracked, and upon completion, results are downloaded to the working directory. By default, the cluster is terminated to avoid incurring unnecessary costs.

3.2. *CDASH* operational sequence

3.2.1. Cluster creation

CDASH adds a new StarCluster template based on the user-defined instance type and size requirements to the StarCluster config file. StarCluster is then used to create a new cluster based on the template. A back-up of the original StarCluster configuration file is created automatically, and the original file is restored once the cluster is in a running state.

3.2.2. File preparation and upload

The *DASH* files are automatically compressed into a *.zip* archive, and the cluster control scripts listed in Table 3 are written to the working directory. Once the cluster is in a running state, *CDASH* uploads all of these files to the master node of the cluster using the StarCluster *put* file transfer tool. The files are then distributed around the cluster *via* a secure copy operation.

3.2.3. Job submission and tracking

The *DASH* jobs are queued for execution using the Sun Grid Engine (SGE) , which is pre-installed on the StarCluster base AMI and which is automatically configured by StarCluster during the initial cluster setup. Jobs are simultaneously executed on all available virtual CPUs (vCPUs) on all nodes of the cluster, including the master node. Job progress is tracked by running the SGE `qstat` command *via* the SSH functionality built into StarCluster. *CDASH* parses the output from this command and provides users with an approximate percentage of the DASH runs completed and an estimate of the time required to complete the remaining jobs.

3.2.4. Result retrieval and summary

Upon job completion, the results of all runs (*.dash* and *.log* result files for each DBF, plus original DBF) are retrieved and collated into a *7zip* archive (Pavlov, 2010) named *results.7z*, which is then downloaded to the working directory. By default, once the results from each worker node have been returned to the master node, the worker node is shutdown to minimise costs. Similarly *CDASH* is by default, set to terminate the cluster once the results have been downloaded to the working directory on the local machine. The files written by *CDASH* (Section 3.2.2, Table 3) are then deleted.

Once *CDASH* has completed the assigned tasks, the user is given a summary containing estimates for the time taken to perform tasks such as cluster start-up and job execution, together with a cost estimate for the cluster. One of the *CDASH* configuration options (Section 3.3) can be enabled to automatically convert the estimate from US$ to a user defined local currency.

3.3. *CDASH* configuration options

CDASH contains a number of user-configurable settings that are summarised in Table 4. These settings provide users with additional options to enable *CDASH* to be customised to their needs. Here, we highlight some of the more important options.

The *7zip* software used to compress the results files may require users to install additional software on their local machine in order to open the results. Users who do not wish to do so may set the `sevenzip` parameter to False. If this is done, results are instead compressed into standard *.zip* archives, which can be opened natively in Windows. As *7zip* archives of DASH result files tend to be *much* smaller than equivalent *.zip* archives, we *strongly* recommend their use to minimise download time. For those who still prefer the use *.zip*, the `downbest` and `numdown` parameters are enabled such that the user can choose to download only a given number of the best results, ranked by their intensity χ^2 values, in order to save time.

The `ID` parameter allows users to create multiple independent clusters simultaneously using *CDASH*. This is of use when a user wishes to spawn several *CDASH*-clusters simultaneously or when multiple users share the same AWS account.

The `convert` Boolean parameter and associated `currency` string allow users not based in the USA to get the cost estimate for the *CDASH* run automatically converted to their local `currency` using an up-to-date currency exchange rate obtained from a call to the Yahoo Finance API (Yahoo, 2015).

The `masternode_different` Boolean parameter (and associated `masternode_type` string) can be used when the desired number of vCPUs is not an integer multiple of the number of vCPUs possessed by the main instance type requested. For example, a cluster with 400 vCPUs could be

obtained using eleven "c4.8xlarge" instances (11 × 36 vCPUs) and one "c4.xlarge" instance (1 × 4 vCPUs).

Amazon splits the EC2 infrastructure into geographically distinct regions. The `switchregion` Boolean parameter is used to toggle the operation of StarCluster in regions other than the default AWS region, which is situated in North Virginia, USA (AWS region = us-east-1). This functionality can be useful for two main reasons:

i) AWS limits the number of instances a user can spawn in a given region. Access to further resources can be obtained by spawning clusters in multiple regions.

ii) If multiple users share the same AWS account, each user could be assigned to a different AWS region ensuring that *CDASH* functionality does not clash with another user.

4. DASH program performance when invoked using *CDASH*

The performance of *CDASH* has been evaluated using two challenging crystal structures; verapamil hydrochloride (VHCl; (Florence *et al.*, 2005)) and ornidazole (ORN; (Shankland & David, 2013)). Structural and data parameters for both materials are listed in Table 5. The experimental parameters used for the *CDASH* runs are listed in Table 6. For comparison purposes, identical sets of *DASH* runs were run on a typical quad-core 3.20 GHz Intel Core i5-4570 standalone Windows PC using *MDASH* to ensure use of all four available processing cores.

The results are presented in Table 7, from which the following important conclusions can be drawn. Firstly, the main overheads associated with using *CDASH* (as opposed to *DASH*) are the cluster start time and network transfer time. The cluster start time increases approximately linearly with the number of instances requested and therefore it is recommended that clusters should be comprised of instance types with the highest number of vCPUs possible in order to reduce this overhead. Network transfer time depends on the network bandwidth available to the user. As mentioned in section 3.3, use of the *7zip* compression software is strongly recommended in order to reduce the overhead associated with the result retrieval. Secondly, the average run time for an individual *DASH* simulated annealing run, running on *c4*-based instances on the EC2 service is approximately double that of the locally run jobs, reflecting the lower performance of each individual vCPU relative to the locally operated CPU cores. Despite this, the ability to leverage very large numbers of vCPUs allows the overall time for execution of a *DASH* job consisting of many simulated annealing runs to be radically reduced.

5. Discussion

In general, as the complexity of a crystal structure increases, the computing power required to solve it by global-optimisation-based SDPD methods increases (Shankland *et al.*, 2013). In some cases, this can become 'rate limiting' with the risk that some structures are then deemed 'too complex to solve'.

The use of cloud resources removes this barrier by enabling access to reliable high-performance computing resources without the need for investment in dedicated hardware that may only be sporadically required for complex cases, or the extensive re-coding required to take advantage of general purpose computing on graphical processing units (GP-GPU acceleration).

The on-demand nature of the EC2 resource also mean that users of *CDASH* do not need to apply for time on, or wait for, high-performance computing resources within their institutions. Instead, a cluster tailored to their specific requirements can be brought online within minutes allowing rapid deployment of SA jobs. Evidently, this performance comes at a cost: time on EC2 is charged on a per-instance-per-hour basis, with time rounded up to the next integer hour. By way of example, a run that takes 61 minutes and one that takes 119 minutes are both charged as two hours of use. This should be considered when planning jobs. Nevertheless, the costs associated with even large *CDASH* jobs are still relatively small compared with those of dedicated hardware (purchase, maintenance and depreciation), especially when the latter is only required periodically.

Other advantages of the cloud-based approach include the ability to tackle large numbers of SDPD jobs in a short period of time. For example, this can facilitate rapid, parallel evaluation of multiple structural input models. Similarly, users can batch process a number of separate *DASH* jobs (*i.e.* sets of SA runs covering different materials) on the same cluster, in order to maximise resource utilisation. Users working on central facility beamlines may benefit from rapid SDPD turnaround, allowing them to modify experiments and recollect data if necessary. Note that CDASH does not have the same hard-coded 999 SA run limit as DASH, and so batches of more than 999 SA runs can be processed. Generating such numbers of SA runs can be accomplished either by using the DASH GUI to generate multiple batches of .DBF run control files, or by using the *dbfgen.py* utility supplied with CDASH.

Whilst EC2 does allow users to run instances based on Windows AMIs, StarCluster does not (at present) support Windows-based clusters. However, given that the effect on the performance of the *DASH* executable when running on Linux virtual machines under Wine is negligible, there is currently no strong imperative to move away from Linux-based clusters.

The current implementation of CDASH involves a certain amount of user intervention in the initial installation phase. Considerable simplification of this process may be achieved by the production of a custom AMI that already incorporates the DASH executable, but this remains to be investigated.

6. Conclusions

We have demonstrated the applicability of infrastructure-as-a-service cloud computing to the problem of SDPD and shown that substantial increases in performance (relative to typically employed local resources) can be obtained by running *DASH* jobs on scalable clusters that are rapidly and easily

created on-demand. We anticipate that this approach will be more attractive to academic and industrial users than the *GDASH* approach.

7. Availability and documentation

The CDASH source code and associated documentation is available online, at https://github.com/mspillman/cdash/. Inherent in the CDASH approach is the use of multiple virtual machines running multiple copies of the DASH executable, and as such, CDASH is only suitable for users with a DASH site license that permits such use. Details of DASH availability can be found at https://www.ccdc.cam.ac.uk/Solutions/PowderDiffraction/Pages/DASH.aspx.

Acknowledgements MJS thanks the University of Reading and the Science and Technology Facilities Council (STFC) for funding. We are grateful to the University of Reading Chemical Analysis Facility for local powder diffraction facilities and to the authors of StarCluster, whose program greatly simplified the task of creating CDASH.

References

Amazon.com Inc. (2010). *Amazon elastic compute cloud (Amazon EC2)*, http://aws.amazon.com/ec2/.

Amazon.com Inc. (2015). *Amazon Web Services EC2 instance types*, http://aws.amazon.com/ec2/instance-types/.

Cianfrocco, M. A. & Leschziner, A. E. (2015). *eLife* **4**.

David, W. I. F., Shankland, K. & Shankland, N. (1998). *Chemical Communications* 931-932.

David, W. I. F., Shankland, K., van de Streek, J., Pidcock, E., Motherwell, W. D. S. & Cole, J. C. (2006). *Journal of Applied Crystallography* **39**, 910-915.

de Oliveira, D., Ocana, K., Ogasawara, E., Dias, J., Baiao, F. & Mattoso, M. (2011). *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pp. 708-715.

Favre-Nicolin, V. & Cerny, R. (2002). *Journal of Applied Crystallography* **35**, 734-743.

Florence, A. J., Shankland, N., Shankland, K., David, W. I. F., Pidcock, E., Xu, X., Johnston, A., Kennedy, A. R., Cox, P. J., Evans, J. S. O., Steele, G., Cosgrove, S. D. & Frampton, C. S. (2005). *Journal of Applied Crystallography* **38**, 249-259.

Griffin, T. A. N., Shankland, K., van de Streek, J. V. & Cole, J. (2009a). *Journal of Applied Crystallography* **42**, 356-359.

Griffin, T. A. N., Shankland, K., van de Streek, J. V. & Cole, J. (2009b). *Journal of Applied Crystallography* **42**, 360-361.

Pavlov, I. (2010). *7-zip file archiver*. http://www.7-zip.org/

Rohlíček, J., Hušák, M. & Favre-Nicolin, V. (2007). *FOX.Grid*, http://fox.vincefn.net/Manual/Fox.Grid

Shankland, K. & David, W. I. F. (2013). Private communication.

Shankland, K., Spillman, M. J., Kabova, E. A., Edgeley, D. S. & Shankland, N. (2013). *Acta Crystallographica Section C* **69**, 1251-1259.

MIT (2014). *StarCluster*, http://star.mit.edu/cluster/

Van Rossum, G. & Drake, F. L. (2003). *Python language reference manual*. Network Theory.

Wine Project (2015). *Wine*. Version 1.6.2.

Yahoo Inc. (2015), *Yahoo Finance*, http://finance.yahoo.com/

Table 1 Latest generation of compute-optimised instance types available on the EC2 service. Instances with the *c4* prefix are based on optimised 2.9 GHz Intel Xeon E5-2666 v3 (Haswell) processors. The virtual CPUs (vCPUs) run as hardware hyperthreads on these processors.

Instance type	vCPU	Memory (GiB)	Cost / hour (US$)*
c4.large	2	3.75	0.110
c4.xlarge	4	7.5	0.220
c4.2xlarge	8	15	0.441
c4.4xlarge	16	30	0.882
c4.8xlarge	36	60	1.763

* As listed on 2015-10-01

Table 2 Local and cloud-based software prerequisites for *CDASH*. It is assumed that the AMI is based on a StarCluster public AMI and hence has all the dependencies necessary for StarCluster use already installed.

Location / OS	Software	Version	Reference
Local / Windows 7	Python 2.7.x	2.7.8	(Van Rossum & Drake, 2003)
	StarCluster	0.95.6	(STAR, 2014)
	DASH	3.3.4	(David *et al.*, 2006)
	7zip (optional)	9.20	(Pavlov, 2010)
Cloud / Ubuntu 14.04	Wine	1.6.2	(Wine Project, 2015)
	p7zip-full	9.20.1	(Pavlov, 2010)
	zip	3.0-8	
	xvfb	2:1.15.1-0ub	
	DASH	3.3.4	(David *et al.*, 2006)

Table 3 Cluster control scripts written by *CDASH*

Script name	Language	Purpose
unzipper.sh	BASH	Unzip *DASH* files
queuejobs.sh	BASH	Submit rundash.sh for every DBF to be processed
rundash.sh	BASH	Run *DASH* in a subdirectory* then copy results into the parent folder
scp.py	Python	Distribute files from master node to worker nodes
getres.py	Python	Retrieve results from all worker nodes and collate them into a single compressed archive (*7zip* or *zip*). By default, terminates worker nodes once results have been retrieved to the master node.

* This is done due to the use of multiple copies of DASH running simultaneously. If all jobs were run in the same directory, there is a risk that errors will arise if the different instances of DASH attempt to read from or write to the same files simultaneously. Such errors are known as *race conditions*.

Table 4 Configuration parameters for CDASH.

Parameter	Data type	Default	Description
`sevenzip`	Boolean	**`True`**	Compress results using the *7zip* software
`downbest`	Boolean	**`False`**	Download best `numdown` results only if `sevenzip` is set to **`False`**
`numdown`	Integer	`20`	Number of results to download if `downbest` is set to **`True`**
`terminator`	Boolean	**`True`**	Terminate cluster automatically
`shutdownnodes`	Boolean	**`True`**	Shutdown worker nodes once jobs have completed
`convert`	Boolean	**`False`**	Convert price estimate to another currency
`currency`	String	`GBP`	ISO 4217 three letter code for currency to convert to if `convert` is set to **`True`**
`verbose`	Boolean	**`False`**	Display output of all commands for debugging
`tracker`	Integer	`30`	Refresh rate of job tracking in seconds
`allowinstances`	csv	*	Comma separated values listing allowed instance types and their cost per hour in $US
`maxnodes`	Integer	`20`†	Maximum number of running instances allowed
`ID`	String	`mycluster`	Cluster identifier
`masternode_different`	Boolean	**`False`**	Allows user to specify a different instance type for the master node. Useful for reaching desired numbers of vCPUs.
`Masternode_type`	String	-	Instance type to set the master node if `masternode_different` is **`True`**
`switchregion`	Boolean	**`False`**	Switch AWS region from default us-east-1
`region`	String	-	Region to start cluster in if `switchregion` is **`True`**

* Comma separated variables which provide a list of the names of the instance types and their associated hourly cost in US$. This items are listed in the order `instance-type, cost-per-hour`. Whilst the default list contains information for all *c3* and *c4* instance types, a truncated example is given below:
`c4.large,0.110,c4.xlarge,0.220,c4.2xlarge,0.441,c4.4xlarge,0.882,c4.8xlarge,1.763`

† AWS limit the number of instances that new users can create in any given region to 20 running instances. Raising this limit is easily accomplished by filling in a request form on the AWS website.

Table 5 Structural and data parameters for verapamil hydrochloride (VHCl) and racemic ornidazole (ORN).

Parameter	VHCl	ORN
a / Å	7.086	13.605
b / Å	10.591	14.054
c / Å	19.196	8.913
α / °	100.10	71.59
β / °	93.73	78.73
γ / °	101.55	64.86
Space group	$P\bar{1}$	$P\bar{1}$
Volume / Å^3	1382.060	1460.086
Z / Z′	2 / 1	6 / 3
CSD reference code	CURHOM	NETRUZ
Degrees of freedom	23*	30
Data source	Laboratory diffractometer	Synchrotron
Radiation	Cu $K_{\alpha 1}$	0.65278 Å
Pawley fit resolution / Å	2.25	2.88
Pawley fit χ^2	2.92	14.95

* The Z-matrix for the verapamil backbone was used without any modification or fixing of automatically detected refinable torsion angles. Normally, the nitrile torsion angle would be fixed resulting in the 22 degrees of freedom reported in previous publications.

Table 6 Experimental parameters for assessing the performance of *CDASH* relative to a typical modern desktop computer.

Parameter	VHCl	ORN
Number of SA runs	108	2160
Number of moves per SA run	2×10^7	2×10^7
EC2 instance type used	c4.8xlarge	c4.8xlarge
Number of instances	3	10
Number of vCPUs	108	360
SA runs per vCPU	1	6

Table 7 Results of the experiments listed in Table 6. Solved = number of SA runs that reached the global minimum; SA run average = average time for each SA run to complete the allotted moves; Total = total time taken; Job = time taken to process the SA runs only; Overhead = time associated with tasks other than SA jobs. For the local runs using *MDASH*, it is assumed that there are no overheads and hence Total = Job. Relative speed is the speed relative to the locally run jobs.

			Time taken / minutes					
Structure	Solved	Environment	SA run avg.	Total	Job	Overhead	Relative speed	Cost / US$
VHCl	2	Local	12.6	331	331	0	1	
		3 × c4.8xlarge	26.8	33.2	27.9	5.3	10	5.29
ORN	3	Local	6.6	3677	3677	0	1	
		10 × c4.8xlarge	14.8	97.4	91.2	6.2	38	35.26

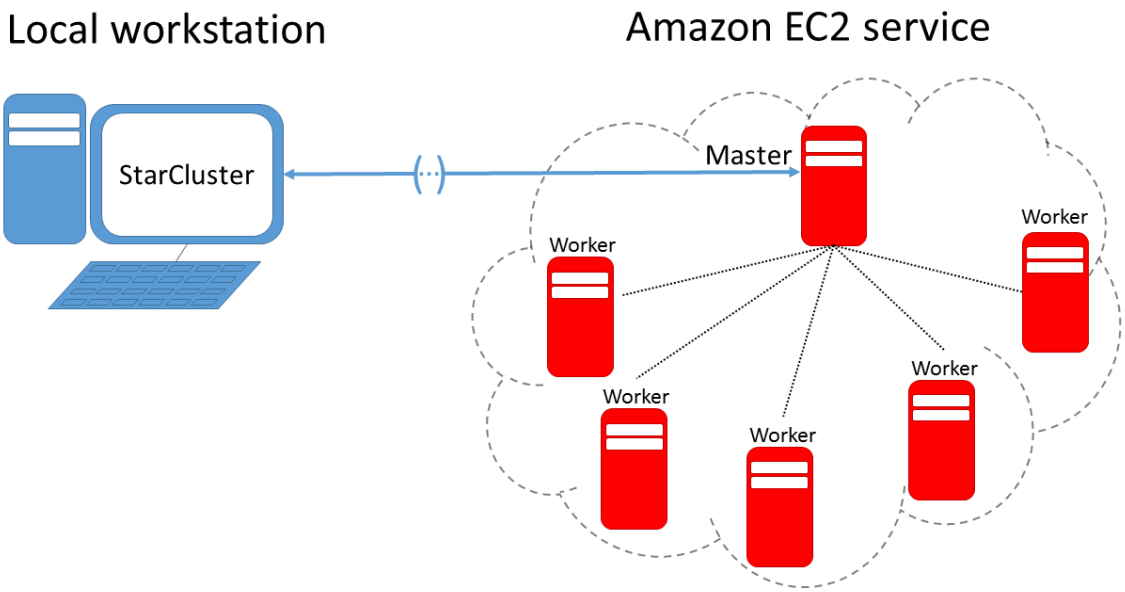

Figure 1 StarCluster running on a local laptop or workstation can be used to automatically build, manage and control clusters located in the cloud. Each instance belonging to a cluster is referred to as a node. Usually, StarCluster will interface directly with a "master node" which then controls the rest of the cluster, or "worker nodes".

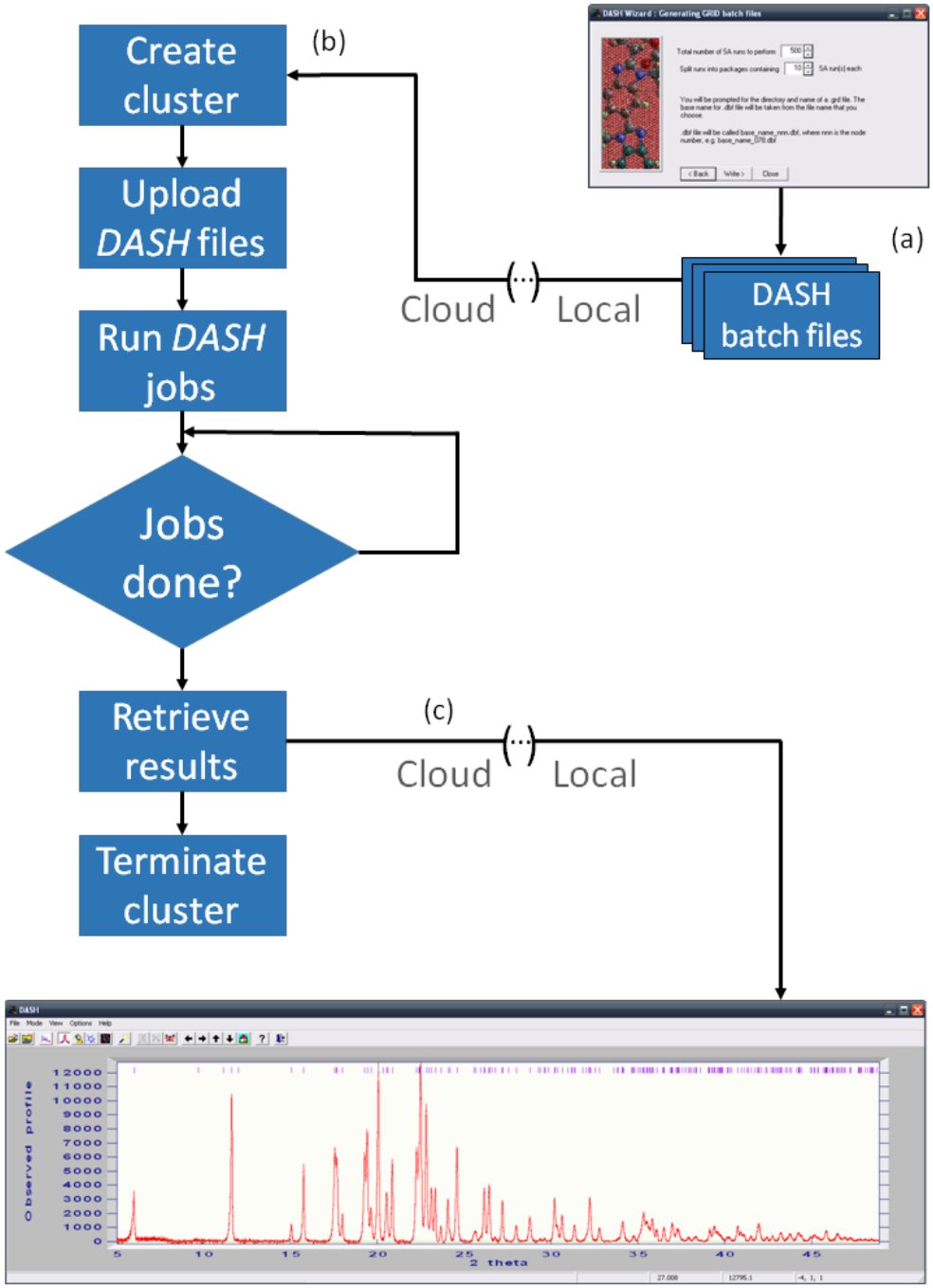

Figure 2 (a) *DASH* batch files are generated within *DASH* following the same procedure as for *GDASH* and *MDASH* (b) *CDASH* creates a cluster with a user-defined number of instances of user specified type and uploads all required files. *CDASH* instructs the cluster to process the DBFs and checks periodically to see if jobs have completed. (c) If so, the results are downloaded from the cluster which is then terminated to avoid incurring unnecessary costs. Results are opened locally in *DASH* as normal.

Supporting information

S1. *DASH* running using Wine

Two locally operated computers with identical hardware running Windows 7 and Ubuntu 14.04 LTS were used to compare the performance of *DASH* running under Windows and Linux (using Wine) environments respectively. Identical *DASH* executables and *DASH* files were used in each case. 50 runs of 2×10^7 SA moves for VHCl were used as a test set. Results are summarised in Table S1. From these results, two important points are clear: (a) the use of Wine has a negligible impact on the performance of *DASH*, with only a few seconds difference between the runs in the different environments and (b) the results obtained in each case were identical and therefore the use of Wine does not affect the accuracy of results obtained by *DASH* when identical *DASH* files are used.

Table S1 Comparison of *DASH* performance when run on Windows and Linux (using Wine) operating systems. Identical *DASH* executables were used, and identical *DASH* run files consisting of 50 SA runs of 2×10^7 SA moves per run on the crystal structure VHCl were processed in each environment.

Parameter	Windows	Linux + Wine
Minimum run time / minutes	15.3	15.4
Maximum run time / minutes	16.6	16.9
Average run time / minutes	16.4	16.6
Minimum profile χ^2	12.28	12.28
Maximum profile χ^2	163.78	163.78
Number of solutions obtained	2	2

S2. *7zip* compression vs *zip* compression for *DASH* result files

The result files of 108 DASH SA runs for VHCl and 2160 DASH SA runs for ORN were compressed using the *7zip* and *zip* file compression packages. The resultant archive file sizes are listed in Table S2. For this particular file type, it is clear that the *7zip* algorithm offers a vastly superior compression ratio and hence its use is strongly recommended.

Table S2 Compression of results from 108 SA runs of VHCl and 2160 SA runs of ORN. The total file sizes are given for the uncompressed data and the resultant *7zip* and *zip* archives.

Results	Uncompressed / MB	*7zip* compression / MB	*zip* compression / MB
VHCl	10.9	0.064	5.44
ORN	245	0.792	123